Vers la synthèse automatique de programmes SIGNAL

Christophe Wolinski, Mohammed Belhadj

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Vers la synthèse automatique de circuits à partir de programmes SIGNAL

Krzysztof WOLINSKI

Mohammed BELHADJ

N° 2039

Septembre 1993

PROGRAMME 2

Calcul symbolique,
programmation
et génie logiciel

ISSN 0249-6399

INRIA
RENNES

Vers la synthèse automatique de circuits à partir de programmes SIGNAL

Krzysztof WOLINSKI
Mohammed BELHADJ

Programme 2 — Calcul symbolique, programmation et génie logiciel
Projet EP-ATR

Rapport de recherche n°2039 — Septembre 1993 — 24 pages

Résumé : Dans ce papier on présente une méthode générale de synthèse d'architecture caractérisée par un contrôle réparti à partir de spécifications faites en SIGNAL. Le résultat final de la synthèse est un réseau de processeurs élémentaires assurant le traitement correspondant aux opérateurs de base du langage SIGNAL, connectés par l'intermédiaire de canaux de communication synchronisés par des événements. Chaque processeur élémentaire possède la même structure composée d'une partie asynchrone utilisant des éléments "delay-insensitive" et une partie synchrone. La partie synchrone est destinée au traitement local tandis que la partie asynchrone assure la synchronisation des calculs.
Les avantages du modèle utilisé sont : l'absence d'une horloge globale (il n'y a pas de problème de "skew"), la séparation des bus pour l'exécution en parallèle des opérations SIGNAL, la vitesse d'exécution dépendant des données (optimisation étant effectuée dynamiquement) et la génération de descriptions structurelles en VHDL.

Mots-clé : synthèse haut niveau, circuits synchrones, circuits asynchrones, SIGNAL

(Abstract: pto)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)

Towards automatic circuits synthesis of SIGNAL programs

Abstract: In this paper we present a general method for architectural synthesis of SIGNAL programs. The generated architecture is charecterized by a distributed control schema. The final result of the synthesis is a net of elemantary processors performing the computation operations corresponding to SIGNAL operator. They are connected by communication hand-shake channels. All elementary processors are composed of two parts: a synchronous and an asynchronous delay-insensitive part one. The former is dedicated to local computation, the later to the synchronization between computations.

The main advantage of the used model are: absence of global clock (no skew problem), separation of buses for parallel execution of SIGNAL operations, data-dependent execution speed using a dynamic optimisation, and generation of structural VHDL descriptions.

Key-words: high level synthesis, synchronous circuit, asynchronous circuit, SIGNAL

Table des matières

1 Introduction

La dernière décennie a connu un accroissement de la capacité des circuits intégrés et de leur vitesse. Les outils de conception déja existant n'arrivaient pas à répondre à ces nouvelles données. Ceci a engendré une recherche active dans le domaine de nouveaux outils de conception. Une des voies qui paraît intéressante est la synthèse automatique d'architectures à partir de spécifications fonctionnelles de haut niveau (voir [7]).

De plus, les architectures asynchrones insensibles aux variations des délais des composants *"delay-insensitive"*, ont connu un regain d'intérêt pour plusieurs raisons [9], parmi lesquelles : l'absence d'horloge globale élimine tous les problèmes de "skew" d'horloge et de "race", les éléments sont facilement composables, la dissipation est très basse, la vitesse dépend de la nature des données, etc.

Mais jusqu'à présent, peu d'outils intégrés à des environements CAO (par exemple l'outil décrit dans [2]) existent et sont utilisés pour la conception de haut niveau de ces architectures.

Dans le domaine des langages synchrones, une approche pour la traduction d'ESTEREL a été entreprise [4]. Cette traduction génère un contrôleur synchrone (avec une horloge globale physique).

Dans la suite on va présenter une approche qui permet de traduire une description fonctionnelle de haut niveau (en SIGNAL) vers une architecture "delay-insensitive". Nous présentons, dans la section suivante le langage SIGNAL, puis la méthode de synthèse est décrite en section trois. Ensuite, un exemple d'application est donné.

2 Le langage SIGNAL

SIGNAL [5][8] est un langage synchrone de type équationnel, construit autour d'un noyau minimum de constructeurs de base. Un programme SIGNAL décrit des relations entre signaux. Un signal est une suite non bornée de valeurs typées ; on associe à chaque signal son horloge qui détermine les instants où les valeurs sont disponibles. Il est possible d'exprimer des contraintes de synchronisation entre les différents signaux, et le compilateur se charge de déterminer si les contraintes sont vérifiées ou non.

Le langage SIGNAL est utilisé pour décrire des applications temps-réel, on peut aussi l'insérer dans une chaîne C.A.O où il permettra à partir d'une spécification de l'algorithme d'aboutir à une mise en œuvre matérielle. Il est muni d'une interface graphique de type bloc-diagramme qui permet une conception aussi bien ascendante que descendante.

SIGNAL utilise l'hypothèse du synchronisme fort, càd que toute action (calcul ou communication) est instantanée. Cette hypothèse a l'avantage de permettre au concepteur de se concentrer sur les aspects logiques du problème sans considération des détails de mise en œuvre (tels que vitesse d'exécution, délais de communication). Ces détails compliquent le problème, et ajoutent des informations dépendant généralement d'un système précis, ce qui rend délicat le transport vers un autre système. Ceci dit, les problèmes de mise en œuvre ne seront pas ignorés, mais reportés à une

phase ultérieure, qui les prendra en considération pour essayer de trouver une mise en œuvre efficace.

Un programme SIGNAL est un ensemble de relations entre signaux, qui spécifient aussi bien les contraintes sur les valeurs que sur les horloges des signaux. Ces contraintes sont exprimées par un ensemble de processus construits avec un ensemble d'opérateurs de base.

2.1 Les opérateurs de base

Il existe deux types d'opérateurs de base en SIGNAL, ceux qui expriment des équations entre signaux synchrones (*expressions fonctionnelles et retard*) et ceux qui composent des signaux d'horloges différentes (*condition et fusion*), ces équations définissent les processus élémentaires.

- **Les expressions fonctionnelles**

  Les fonctions définies sur les types du langage (par exemple les fonctions arithmétiques ou booléennes), sont étendues canoniquement aux signaux. Le signal (Y_t) défini par :

  $$Y_t = f(X1_t, X2_t, ..., Xn_t) \quad t \in T$$

  où f est une fonction instantanée, s'écrit en SIGNAL :

  Y :=f(X1,X2,...,Xn)

  Les signaux Y,X1,X2,...,Xn ayant le même ensemble d'indices ont la même horloge (ils sont synchrones).

- **Le retard**

  Le retard fournit les valeurs passées d'un signal, on note généralement :

  $$ZX_t = X_{t-1} \quad t \in T$$

  qui s'écrit en SIGNAL :

  ZX := X $ 1

  Les signaux X, ZX sont synchrones, càd disponibles au même instant, et ZX contient la valeur précédente de X. Pour l'instant initial ZX contient une valeur v_0 spécifiée par la déclaration suivante :

  ZX **init** v_0

- **Les expressions conditionnelles**

  On peut sous-échantillonner un signal par une condition booléenne :

  Y := X **when** C

Y est présent quand X est présent et C est présent avec la valeur *vrai*. Quand Y est présent, sa valeur est égale à celle de X. Y est à la fois moins fréquent que X et que C.

Exemple :

X :	x_1	x_2	$\perp$	x_3	$\perp$	x_4	x_5	x_6	...
C :	F	V	F	F	V	V	$\perp$	V	...
Y :	$\perp$	x_2	$\perp$	$\perp$	$\perp$	x_4	$\perp$	x_6	...

le symbole $\perp$ représente l'absence de valeur.

- **La fusion**

  Cet opérateur permet de fusionner, de manière déterministe, deux signaux de même type :

  Z := X **default** Y

  La valeur de Z est égale à la valeur de X quand X est présent, sinon, égale à la valeur de Y quand Y est présent et X est absent. Z est donc plus fréquent que X et que Y.

  Exemple :

X :	x_1	x_2	$\perp$	x_3	$\perp$	...
Y :	z_1	$\perp$	z_2	$\perp$	z_3	...
Z :	x_1	x_2	z_2	x_3	z_3	...

2.2 La composition de processus

Les processus élémentaires peuvent être composés à l'aide d'un opérateur associatif et commutatif, noté "|", qui dénote l'union de systèmes d'équations. Par exemple, l'équation $x_t = x_{t-1} + y_{t-1}$ peut être récrite en $\{x_t = zx_t + zy_t, zx_t = x_{t-1}, zy_t = y_{t-1}\}$ qui se traduit en SIGNAL en le processus suivant (|x :=zx+zy |zx :=x$ 1|zy :=y$1|). On peut appliquer cette opération récursivement sur les processus ainsi construits. Le principe de l'assignation unique doit être respecté pour que les processus soient définis. Le confinement d'un signal x à un sous-processus P est possible par l'opérateur d'**oblitération** noté P/x.

2.3 Modularité et processus dérivés

Un processus peut être associé à une définition de schéma qui fournit un mécanisme de structuration de programme. Un tel schéma (cf. ci-dessous) est formé d'un nom (VAR ci-dessous), d'une liste de paramètres typés, d'une liste de signaux d'entrées typés (précédée du caractère "?"), d'une liste de signaux de sortie typés (précédée du caractère "!") d'un processus et de déclarations locales précédées du mot clé **where**.

```
process VAR = ( integer V0)
              {? integer IN ! integer OUT}

              (| CV  := IN default ZCV
               | ZCV := CV$1
               | OUT := CV when event OUT      % event est de'crit ci-dessous %
               |)
              where integer CV, ZCV init V0
end
```

Le schéma peut contenir, dans sa zone de déclarations locales, la déclaration de sous-schémas.

Une occurence d'un schéma P dans le texte d'un processus y provoque la duplication et l'insertion du système qu'il dénote (on associe à chaque processus SIGNAL un système d'équations) dans le contexte de cette occurrence.

A l'aide des processus élémentaires présentés plus haut, et de l'opérateur de composition, certains processus dérivés ont été intégrés au langage pour permettre des descriptions plus concises de programmes SIGNAL. On peut en citer :

- **synchro** $\{ x_1, x_2, .., x_n\}$ qui spécifie la synchronisation explicite des signaux $x_1, x_2, .., x_n$.
- h :=**event** x, h est l'horloge de x, càd que h est le signal booléen toujours à vrai à la fréquence de x.
- **when** c est équivalent à c **when** c.
- X **cell** B est une mémorisation du signal X quand B est vrai.

Des tableaux de signaux et de processus ont été introduits. Il sont utiles dans les programmes systoliques et les programmes de contrôle. Pour plus de détails voir [5].

2.4 Codage temporel d'un processus SIGNAL

SIGNAL utilise un codage dans $\mathcal{Z}/3\mathcal{Z}$, pour représenter l'absence, la présence avec la valeur *vrai*, la présence avec la valeur *faux* d'un signal, que l'on code de la manière suivante :

- l'absence d'un signal est codée par 0.
- la présence d'un signal booléen avec la valeur *vrai* ou la présence d'un signal non booléen est codée par 1.
- la présence d'un signal booléen avec la valeur *faux* est codée par -1.

Pour un signal x son horloge est codée par x^2 dans le corps $\mathcal{Z}/3\mathcal{Z}$.On obtient, pour les opérateurs de base, les équations d'horloges suivantes :

- $x := f(x_1, .., x_n) \longrightarrow x^2 = x_1^2 = ... = x_n^2$
- $x := a\$1$ **init** $v \longrightarrow x^2 = a^2$

- $x := a$ **when** $c \longrightarrow x^2 = a^2(-c - c^2)$
- $x := a$ **default** $b \longrightarrow x^2 = a^2 + (1 - a^2)b^2$

Un cas particulier se présente quand un signal booléen est défini par un retard : On fait référence à la valeur passée du signal. Ceci conduit à l'utilisation d'un système dynamique [3]. On a donc pour le processus $x := a\$1$ **init** v (avec x booléen) le système suivant :

$$\xi' = a + (1 - a^2)\xi, \xi_0 = v$$

$$x = a^2\xi$$

où ξ' représente l'état courant du système, ξ, l'état précédent et ξ_0 l'état initial. L'état courant change si et seulement si le signal a est présent, sinon il conserve son ancienne valeur ξ.

En utilisant ce codage, on associe à chaque programme SIGNAL un système d'équations, et on peut ainsi analyser le comportement temporel du programme et détecter d'éventuelles incohérences temporelles. Cette analyse est nommée **calcul d'horloge**. Si le calcul d'horloge ne fait pas intervenir les systèmes dynamiques il est appelé **calcul statique**, et dans le cas contraire **calcul dynamique**.

2.5 Graphe aux Dépendances Conditionnées

Pour chaque processus de base, on définit les dépendances de données qui lui sont associées. On représente une dépendance par une flèche conditionnée par l'horloge à laquelle la dépendance est effective.

Nous associons donc à chaque processus élémentaire (correspondant à un opérateur de base) les dépendances suivantes :

- $x := f(x_1, .., x_n) : \forall i \in [1..n], x_i \xrightarrow{x^2} x$

  La sortie x dépend de toutes les entrées x_i à chaque instant où elles sont définies ($x^2 = 1$).

- $x := a\$1$

  il n'y a pas de dépendance entre x et a autre que la causalité temporelle implicite

- $x := a$ **when** $c : a \xrightarrow{x^2} x$

  x dépend de a quand a est présent ($a^2 = 1$) et c présent avec la valeur *vrai* ($-c - c^2$).

- $x := a$ **default** $b : a \xrightarrow{a^2} x \xleftarrow{(1-a^2)b^2} b$

  x dépend de a lorsque a est présent (a^2), et dépend de b lorsque a est absent et b est présent $(1 - a^2)b^2$.

En outre, chaque signal dépend de son horloge : $x^2 \xrightarrow{x^2} x$. Pour de plus amples détails, voir [6]. A chaque instruction SIGNAL, un graphe de dépendances conditionnées est donc associé. Le graphe correspondant au programme SIGNAL est formé par la composition des graphes associés à chacune de ses instructions. Ce graphe (noté **GDC**) est construit à la compilation, il permet entre autres de vérifier l'absence de cycle de dépendances. Il est aussi utilisé pour d'autres tâches : génération de code, optimisations et partitionnement, répartion et placement de programmes SIGNAL dans une machine multiprocesseurs [6].

3 Les éléments "delay-insensitive"

Un circuit est dit *delay-insensitive* (ou *self-timed*) s'il fonctionne correctement, et ceci indépendamment de toute hypothèse sur les délais dans les fils et les opérateurs, à condition qu'ils soient bornés. Ces circuits n'utilisent pas de signal d'horloge. Le séquencement est entièrement forcé par les méchanismes de communication. Dans notre travail, on va utiliser les composants suivants (pour plus de détails voir [9]) :

Elément c-muller.

Réalise une fonction AND sur les événements. Un événement est une transition, càd un front montant ou descendant. La description est donné en SIGNAL (une description en VHDL est donnée dans [10]) :

```
process C_MULLER=
         (integer DELAI)
         {? logical X,Y
          ! logical Z
           }
         (| ETAT := X when (X=Y) default ZETAT
          | ZETAT:= ETAT$1
          | Z:= ETAT $ DELAI
          | synchro{X,Y,ETAT} |)/ETAT,ZETAT
         where
          logical ETAT,ZETAT init false
end
```

On note içi que le retard SIGNAL utilisé et le délai de transport. Pour le délai inertiel il suffit de définir un processus INERT_DELAI, nous l'omettons pour des raisons de clarté en sachant qu'il toujours possible de substitué le retard "$" avec la définition du processus INERT_DELAI.

Elément select.

Dirige l'événement de l'entrée x vers la sortie s_v (resp. s_f) si la valeur du signal sel est vraie (resp. faux) :

```
process SELECT =
     (integer DELAI)
     {? logical X;
        logical SEL
      ! logical S_V,S_F}
```

```
     (| S_F := ETAT_F $ DELAI
      | S_V := ETAT_V $ DELAI
      | ETAT_V := (not ZETAT_V when SEL) default ZETAT_V
      | ZETAT_V := ETAT_V $ 1
      | ETAT_F := (not ZETAT_F when (not SEL)) default ZETAT_F
      | ZETAT_F := ETAT_F $ 1
      |)
    where
      logical ETAT_V,ETAT_F, ZETAT_V init true, ZETAT_F init true
end
```

Elément toggle.

Dirige l'événement de l'entrée x alternativement vers la sortie sp, puis p en commencant par sp :

```
process TOGGLE=
        (integer DELAI)
         {? logical X
         ! logical S_P, S
           }
         (| JETON := (not ZJETON when X_EVENT) default ZJETON
          | ZJETON := JETON $1
          | X_EVENT := not(X=ZX)
          | ZX     := X$1
          | ETAT   := (not ZETAT when(JETON and ZX) )default ZETAT
          | ZETAT := ETAT $ 1
          | ETAT_P := (not ZETAT_P when(not JETON and ZX) )
                      default ZETAT_P
          | ZETAT_P:= ETAT_P $ 1
          | S_P    := ETAT_P $ DELAI
          | S      := ETAT $ DELAI
          |)
        where
          logical JETON, ETAT_P, ETAT, ZJETON init false,
          ZETAT_P init true, ZETAT init true, ZX init false
end
```

Elément ⊕.

Réalise une fonction OR sur les événements :

```
process XOR=
        (integer DELAI)
         {? logical A,B
         ! logical C
          }
         (| CC := (A and (not B)) or (B and (not A))
          | C := CC $ DELAI |)
         where
          logical CC init false
end
```

4 La synthèse

La synthèse des circuits exécutant les programmes SIGNAL repose sur les propriétés algébriques des opérateurs du langage. Elle a pour support le graphe **GDC** introduit ci-dessus.

La synthèse est effectuée en deux étapes :

- la première étape est la construction d'un réseau (qui traduit le **GDC** d'un programme) composé des éléments suivants :
  - c_cell correspondant au CELL
  - c_when correspondant au WHEN
  - c_rwhen correspondant au WHEN à une entrée
  - c_default correspondant au DEFAULT
  - c_$ correspondant au $
  - c_com correspondant à l'appel de procédure
  - c_op correspondant à un opérateur arithmétique/logique
  - fork correspondant à la diffusion d'un signal vers plusieurs opérateurs
- la deuxième étape consiste à réduire certaines parties du réseau (les parties de réseau délimitées par les éléments: fork, c_when, c_cell, c_default) créées dans la première étape de la synthèse en les remplacant par des processeurs élémentaires.

Le résultat final des deux transformations successives est un réseau composé de processeurs élémentaires et d'opérateurs "fork" connectés par des canaux synchronisés par des événements où un événement est un front montant ou un front descendant. Dans le réseau obtenu, chaque instruction prend un temps non nul pour s'exécuter. La difficulté de la mise en œuvre provient de cette désynchronisation des événements logiques instantanés. Nous donnons dans la suite un schéma de traduction informel dans le langage VHDL.

4.1 Les canaux

Chaque canal "c", connectant les nœuds du réseau est composé des quatre éléments $\{c_v, c_{hv}, c_h, c_{hacq}\}$ où :

- c_v - est une donnée correspondant à une valeur du signal "c" du langage SIGNAL
- c_{hv} - est la valeur d'horloge du signal "c". Si $c_{hv} = 1$ alors l'horloge du signal "c" est présente pour l'instant actuel de l'horloge "c_h"
- c_h - est l'horloge la plus rapide pour cette partie du réseau (physiquement elle est représentée par un événement)
- c_{hacq} - est un acquittement signalant la fin du traitement correspondant à l'événement "c_h".

Exemple de canal :